International Journal of Neural Systems, Vol. 0, No. 0 (2017) 1–19

Simplified and yet Turing universal spiking neural P systems with communication on request

Tingfang Wu
Key Laboratory of Image Information Processing and Intelligent Control of Education Ministry of China, School of Automation, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China

Florin-Daniel Bîlbîe
Department of Computer Science, Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei nr.14, sector 1, C.P. 010014, Bucureşti, România

Andrei Păun
Department of Computer Science, Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei nr.14, sector 1, C.P. 010014, Bucureşti, România
Bioinformatics Department, National Institute of Research and Development for Biological Sciences, Splaiul Independenţei, Nr. 296, Sector 6, Bucureşti, România

Linqiang Pan*
Key Laboratory of Image Information Processing and Intelligent Control of Education Ministry of China, School of Automation, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China
School of Electric and Information Engineering, Zhengzhou University of Light Industry, Zhengzhou 450002, Henan, China

Ferrante Neri
Centre for Computational Intelligence, School of Computer Science and Informatics, De Montfort University, The Gateway, Leicester LE1 9BH, England, United Kingdom

Spiking neural P systems are a class of third generation neural networks belonging to the framework of membrane computing. Spiking neural P systems with communication on request (SNQ P systems) are a type of spiking neural P system where the spikes are requested from neighbouring neurons. SNQ P systems have previously been proved to be universal (computationally equivalent to Turing machines) when two types of spikes are considered.
This paper studies a simplified version of SNQ P systems, i.e. SNQ P systems with one type of spike. It is proved that one type of spike is enough to guarantee the Turing universality of SNQ P systems. Theoretical results are shown in the cases of the SNQ P system used in both generating and accepting modes. Furthermore, the influence of the number of unbounded neurons (the number of spikes in a neuron is not bounded) on the computation power of SNQ P systems with one type of spike is investigated. It is found that SNQ P systems functioning as number generating devices with one type of spike and four unbounded neurons are Turing universal.

Keywords: Bio-inspired computing; Membrane computing; Spiking neural network; Spiking neural P system; Computation power

*Corresponding author: lqpan@mail.hust.edu.cn; lqpanhust@gmail.com

1. Introduction

Neural network models[1,2] have been extensively researched and have successfully been applied in various fields, including pattern recognition,[3–5] optimization problems,[6,7] modelling,[8,9] time series prediction,[10] gesture recognition,[11] and machine learning.[12–14] Spiking neural networks[15,16] (SNNs, for short) are neural network which make use of electrical impulses, called *spiking rules*, as activation functions. As third generation neural networks, spiking neural networks have gained their popularity thanks to their learning capability[17–21] and their great potential for solving some complicated time-dependent problems.[22–25] Some examples of spiking neural network applications are in medical diagnostics,[26] epilepsy examination,[23,27,28] neurosurgery,[29] pattern recognition,[30,31] information processing,[32] and liquid-state machine circuitry.[33]

Spiking neural P systems[34,35] (abbreviated as SN P systems) are a class of spiking neural networks belonging to the field of *membrane computing*. SN P systems incorporate the idea of spiking neurons into membrane computing models.[36–38]

An SN P system is characterised by the following ingredients composing it:

- *neurons* which contain a number of spikes representing the neuron's state;
- *synapses* that connect the neurons to form a *directed graph structure*, where the nodes represent neurons and the arcs represent synapses;
- *rules* (*spiking rules* and *forgetting rules*), a kind of integrate-and-fire functions in the form of a formal grammar production, which evolve spikes for updating the states of neurons.

The functioning of an SN P system can be summarised in the following way. Neurons in the system cannot fire at each step until the number of spikes inside the neuron reaches a specific value. When a neuron fires by using a spiking rule, it consumes a specified number of spikes and generates a specified number of spikes which are sent to the neurons connected by synapses with the neuron where the rule is applied; while by using a forgetting rule, it just consumes a predefined number of spikes. The outgoing spike train emitted by the output neuron can be interpreted as different numbers in various methods to encode information, e.g., the frequency of spikes[39,40] and the timing between spikes,[34,41] stochastic delays of spiking rules and stochastic loss of spikes.[42] A SN P System with a probabilistically established firing time has been proposed in Ref. 43.

Theoretical aspects of SN P systems have been investigated after their introduction.[34] These studies analyse the computation power of SN P systems as various computing devices. Some examples include number generating devices,[34,44] where the output neuron outputs the generated numbers encoded by spikes; number accepting devices,[34] where the input neuron receives the numbers encoded by spikes from the environment; language generators,[45] where the output neuron emits a number of spikes associated with a symbol in each step and the sequences of these symbols associated with halting computations constitute a language. SN P systems used as these computing devices were proved to be equivalent to Turing machines.

The computation efficiency of SN P systems has also been studied. It was shown that SN P systems are capable of solving computationally hard problems in a polynomial or linear time under certain conditions, e.g., by using pre-computed resources or by using neuron division or neuron budding.[46–48]

Theoretical and practical applications of SN P systems have been researched, covering e.g. image processing,[49] fault diagnosis,[50–52] circuit simulation and implementation,[53–55] integrated circuit design/configuration,[56,57] optimization problems.[35] A study on simulators for SN P Systems is reported in Ref. 58.

In the above mentioned SN P systems, neurons communicate on command pattern, in the sense that the emitting neuron emits spikes to all the connected neurons along its synapses. In this way, the emitting neuron has the initiative for communication in terms of spikes exchange pattern. Inspired by the way that components communicate with each other by a request-response pattern in parallel-cooperating grammar systems,[59,60] a novel communication strategy between neurons, called communication on request, was proposed and developed in SN P systems.[61] The resulting models are called spiking neural P systems with communication on request (SNQ P systems). In these models, spikes are transmitted from one neuron to another one only when the re-

ceiving neuron makes a query to request spikes from the emitting neuron.

SNQ P systems, like neural networks which can have multiple activation functions, can have multiple types of spikes. When SNQ P systems were introduced,[61] it was shown that systems with two types of spikes are Turing universal (computationally equivalent to Turing machines).

In this work, with the aim of proposing a simplified and equally powerful computational model, we investigate the computation power of SNQ P systems with one type of spike. It is proved that one type of spike is sufficient for SNQ P systems achieving Turing universality as both number generating and number accepting devices.

Furthermore, the influence of the number of unbounded neurons (those neurons which do not have a maximum number of spikes) on the computation power of SNQ P systems is investigated. It is proved that SNQ P systems with four unbounded neurons are also Turing universal.

In order to better highlight the motivation of this article, it is worth mentioning that the integrate-and-fire condition in the form of regular expressions is computationally expensive, see Ref. 62. More specifically, to decide whether or not a number is in the set of numbers associated with a regular expression is an **NP**-complete problem.[63] The complexity of problems associated to regular expressions and techniques to reduce the complexity and/or simplify the task are topics explored in the neural system literature for classical SN P systems. For example, in Ref. 64, a restriction of the form of regular expressions was proposed, in other works regular expressions were not used at all and replaced with the potential of a neuron and a threshold[65] or with polarizations associated with neurons.[66] Thus, the search of computationally cheap or efficient integration-and-fire conditions, such as resulting from reduction to one type of spike, is of interest to both theoretical and applied research.

The remainder of this paper is organized as follows. Section 2 recalls basic concepts needed in the following sections and gives the formal definition of SNQ P systems with one type of spike. The results about the Turing universality of SNQ P systems as number generators and acceptors are presented in Sections 3 and 4, respectively. Finally, conclusions are given in Section 5.

2. SNQ P systems with one type of spike: definitions and generalities

This section provides basic definitions and introduces the notation used in this work. Furthermore, this section briefly describes the functioning of SNQ P systems with one type of spike.

Since only one type of spike is used in this paper, we focus on the SNQ P systems with only one type of spike. More details about the definition of classical SN P systems and SNQ P systems are available in Refs. 61, 67. Some notions from formal language and automata theory are used in the definition of SNQ P systems, including the notion of regular expression; for the detail, the reader is referred to Refs. 68, 69.

Definition 1. A Spiking Neural P system with communication on request (SNQ P system) with one type of spike, of $m \geq 1$ neurons, is a tuple

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, in, out)$$

where:

- $O = \{a\}$ is the singleton alphabet, where a denotes the only type of *spike* under consideration;
- $\sigma_i = (n_i, R_i)$ with $1 \leq i \leq m$ are the *neurons* composing the SNQ P system, where:

(1) $n_i \geq 0$ is *the initial number of spikes* present in neuron σ_i;

(2) R_i is a finite set of rules of the form E/Qw, where E is a regular expression over the empty string λ or the singleton alphabet $\{a\}$, Q means requesting spikes, and w is a finite non-empty sequence of queries of the forms $w = (a^p, j)$ or $w = (a^\infty, j)$, with $p \geq 0$, $1 \leq j \leq m$, and $i \neq j$;

- $in, out \in \{1, 2, \ldots, m\}$ indicate the *input and output neurons* of the system, respectively.

Before introducing the work of SNQ P systems, we recall the notion of regular expression.[68,69] Given an alphabet V, a regular expression E is defined as follows: (i)] the empty string λ is a regular expression; (ii) each symbol $a \in V$ is a regular expression; (iii) if E_1 and E_2 are two regular expressions over V, then the union and concatenation of (E_1) and (E_2), written $E_1 \cup E_2$ and $(E_1)(E_2)$, are regular expressions over V; (iv) if E is a regular expression, then

the Kleene star of E, written E^+, is a regular expression over V. In this above definition of SNQ P systems, the regular expression E is defined on the singleton alphabet $\{a\}$.

To better explain the notation as well as the functioning of SNQ P systems with one type of spike, let us consider a neuron σ_i in the system. For exemplification purposes, assume that the neuron σ_i contains a rule $E/Q(a^p, j)$ (in this case $w = (a^p, j)$). Let us focus, at first, on the fact that j appears in the rule. This rule implicitly encodes some pieces of information about the topology of the SNQ P system as it implicates that there exists a synapse (j, i) connecting neuron σ_j to neuron σ_i. Let us focus on the a^p part of the rule. The statement a^p means that p copies of the spike type a are requested. Thus, the query of the form (a^p, j) in a rule in neuron σ_i means that neuron σ_i requests p copies of the spike a from neuron σ_j. The notation (a^∞, j) refers to the degenerate case, i.e., it means that neuron σ_i requests all spikes present in neuron σ_j.

A rule $E/Q(a^p, j)$ in neuron σ_i is enabled only if the following two firing conditions are satisfied: (1) the neuron σ_i must have the capability to contain p spikes, i.e., the content of neuron σ_i is described by the regular expression E; when the regular expression E is the empty string λ, then the neuron must be empty; (2) the neuron σ_j must contain not less than p copies of spikes.

When two or more neurons request an identical number of spikes from the same neuron σ_j, that is the rule (a^p, j) appears on multiple neurons, only p copies of spikes are removed from the emitting neuron σ_j, and these p spikes are replicated and transmitted to each of the requesting neurons.

When two or more neurons request spikes from the same neuron, but with different numbers of spikes, i.e. two queries (a^p, j) and (a^q, j) with $p \neq q$ in a rule in neuron σ_i are fired from two different neurons, then these rules cannot be applied simultaneously. When this situation occurs it is said that the neurons perform *conflicting queries*. In the case of conflicting queries one spike request (and thus the requesting neuron) is non-deterministically selected while the other request is ignored.

An SNQ P system starts from its *initial configuration* which is described by the number of spikes present in every neuron, and proceeds by applying the rules as described above, then a *transition* of the system from a configuration to the next one is obtained. A *computation* is a (finite or infinite) sequence of transitions starting from the initial configuration such that each non-first term of the sequence is obtained from the previous configuration by applying the rules as described above; if the sequence is finite (called *halting computation*) then the last term of the sequence is a *halting configuration*, that is, a configuration where no rule of the system is applicable to it.

A *halting computation* means that the SNQ P system reached a configuration where no rule can be longer applied in all the neurons. The *result of a computation* is defined as the total number of spikes present in the output neuron when the computation halts.

An SNQ P system Π can work as a number generator/acceptor, that is, it can be used to either generate or accept a number n. In the case of Π working as a generator, it starts from its initial configuration and eventually halts. When the system halts, n spikes are present in the output neuron σ_{out}. The set of numbers generated in this way by system Π is denoted by $N_{all}(\Pi)$. In the accepting mode, a number n is introduced into Π through the input neuron σ_{in}. If the system halts, then the number n is accepted by the system. The set of numbers accepted by system Π is denoted by $N_{acc}(\Pi)$.

In what follows, we give an example to show how an SNQ P system works with the aim of illustrating the aforementioned notions. The system Π given in Figure 1 consists of three neurons with labels 1, 2, and *out*, respectively. Initially, neuron σ_{out} contains two spikes, and neurons σ_1 and σ_2 have no spike.

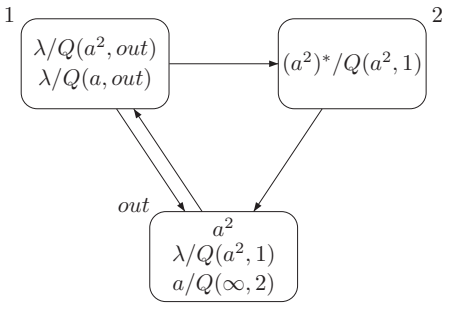

Figure 1. An SNQ P system Π.

System Π works as follows. At step 1, rules $\lambda/Q(a^2, out)$ and $\lambda/Q(a, out)$ in neuron σ_1 are enabled, since the firing conditions of the two rules are

satisfied: neuron σ_1 contains no spike, which satisfies the firing condition (1), and neuron σ_{out} contains two spike, which satisfies the firing condition (2). Thus, one of these two rules is applied, nondeterministically chosen. If rule $\lambda/Q(a^2, out)$ is applied, then neuron σ_1 requests two spikes from neuron σ_{out}. At step 2, rule $\lambda/Q(a^2, 1)$ in neuron σ_{out} and rule $(a^2)^*/Q(a^2, 1)$ in neuron σ_2 are enabled, because neurons σ_{out} and σ_2 are empty, and neuron σ_1 contains two spikes, which are not less than the requested spikes, so both rules are applied, each neuron requesting two spikes from neuron σ_1. In this way, as long as rule $\lambda/Q(a^2, out)$ is chosen to be applied, neurons σ_1 and σ_{out} can alternately request two spikes from each other; at the same time, the number of spikes in neuron σ_2 is increased by two during this process.

If at some step t (it is possible at the first step), rule $\lambda/Q(a, out)$ is chosen to be applied, then neuron σ_1 requests one spike from neuron σ_{out}. In this case, neuron σ_1 contains one spike, but one of the firing conditions of this rule is that neuron σ_1 contains no spike, so no rule inside neuron σ_1 are enabled; and due to the lack of spikes, both neurons σ_{out} and σ_2 cannot request spikes from it. At step $t + 1$, neuron σ_{out} contains one spike inside, thus the firing condition of rule $a/Q(\infty, 2)$ is satisfied, and the rule is applied, then neuron σ_{out} requests all spikes of neuron σ_2 which contains an even number of spikes. After this step, the computation of Π halts. In this way, the number of spikes in neuron σ_{out} becomes odd. Therefore, according to the definition of the computation result, the set of numbers generated by system Π is $N_{all}(\Pi) = \{2n + 1 \mid n \geq 0\}$.

Definition 2. In an SN P system, a neuron is said to be *bounded* if a bound is imposed on the number of spikes in the neuron (to any computation of the system), whereas a neuron is said to be *unbounded* if the number of spikes inside it is not bounded.[34] An SN P system is said to be *bounded* if all neurons are bounded in the system; an SN P system is said to be *unbounded* if it contains at least one unbounded neuron.

Definition 3. The family of all sets $N_{all}(\Pi)$ (resp., $N_{acc}(\Pi)$) generated (resp., accepted) by SNQ P systems of degree m is indicated as $N_{all}SN_kQP_m(unb_s)$ (resp., $N_{acc}SN_kQP_m(unb_s)$), where the subscript *all* indicates that the computation result is encoded by the number of spikes present in the output neuron when the computation halts (resp., the subscript *acc* indicates systems work in the accepting mode), k indicates the maximal number of types of spikes, and s indicates the maximal number of unbounded neurons. If one of the parameters k, m, and s is not bounded, then it is replaced with $*$.

Definition 4. A number is said *Turing computable* when they are calculable by finite means.[70] The family of Turing computable sets of numbers is indicated with NRE.

When a type of computing devices is conceptually proposed, a fundamental task is to show their equivalence with Turing machines, since it would guarantee the coherence and functionality of the proposed models. In order to perform this task, a popular way is to prove that a proposed computing device can be used as a number generator and the set of the generated numbers coincides with the set of numbers generated by the corresponding Turing machine. If the proposed computing devices are equivalent to Turing machines, then they are said *Turing universal*.

The Turing universality of SNQ P systems has been proved by showing that the family of sets of numbers generated by SNQ P systems with at most two types of spikes is equivalent to the family of sets of Turing computable numbers NRE,[61] where the number of neurons and the number of unbounded neurons in these SNQ P systems are not bounded. By following the notation introduced above, this statement can be expressed as $N_{all}SN_2QP_*(unb_*) = NRE$.

3. Universality of SNQ P systems with one type of spike working as number generators

This section extends the previously achieved theoretical result[61] and shows the universality of SNQ P systems with only one type of spike. These results allow the conceptual formulation of a very simple and yet powerful computational device.

Theorem 1. *SNQ P systems with one type of spike are Turing universal:*

$$N_{all}SN_1QP_*(unb_*) = NRE.$$

Proof. The inclusion $N_{all}SN_1QP_*(unb_*) \subseteq NRE$ can be obtained from the Turing-Church thesis.

Thus we only have to prove the inclusion $NRE \subseteq N_{all}SN_1QP_*(unb_*)$. The following proof is based on the simulation of nondeterministic register machines working in the generative way. As we know, register machines can characterize NRE[71] (the family of recursively enumerable sets of numbers, i.e., the length sets of recursively enumerable languages). Here, we briefly recall the definition of register machines. A non-deterministic register machine is of the form $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halt label (in all the neurons no rule can be longer applied), and I is a finite set of instructions bijectively labelled by elements from H. The instructions are of the following forms:

- $l_i : (\texttt{ADD}(r), l_j, l_k)$ (ADD instruction)
  Add 1 to the content of register r and go to one of the instructions with labels l_j and l_k, nondeterministically chosen.
- $l_i : (\texttt{SUB}(r), l_j, l_k)$ (SUB instruction)
  If register r is not empty, then subtract 1 from its content and go to the instruction with label l_j; otherwise, go to the instruction with label l_k.
- $l_h : \texttt{HALT}$ (the halt instruction)
  No rule can be applied in the neurons. The halt label l_h is only assigned to this instruction.

The set of all numbers generated by the nondeterministic register machine M is denoted by $N(M)$. For more information about register machines, the reader is referred to Refs. 67, 72.

Consider a register machine $M=(m, H, l_0, l_h, I)$ with the assumption that register 1 stores the result of the computation, which is never a subject of subtraction operations. It is also assumed that l_0 labels an ADD instruction and all registers different from register 1 are empty in the halting configuration. In what follows, in order to simulate machine M, we construct a specific SNQ P system Π with one type of spike.

In system Π, there are two types of modules: the ADD and SUB modules, which are used to simulate the ADD and SUB instructions of M, respectively. Unlike traditional SN P systems, in system Π, there is not an OUTPUT module to output the computation result, since the result of a computation is defined as the number of spikes present in a specified neuron at the moment when the computation halts.

In general, each register r of M corresponds to a neuron σ_r in system Π. The value stored in register r is equal to the number of spikes present in neuron σ_r, i.e., if register r contains a number n, then neuron σ_r contains n copies of spikes. In M, each label $l_i \in H$ of an instruction also corresponds to a neuron σ_{l_i} as well as some auxiliary neurons $\sigma_{l_i^{(j)}}$, $j = 1, 2, 3, \cdots$. Note that each label $l_i \in H$ labels a unique instruction of M, so all neurons σ_{l_i}, $\sigma_{l_i^{(j)}}$ are also precisely associated with a unique instruction of M. There is a distinguished neuron σ_g in system Π, working like a "garbage collector" to remove the redundant spikes in some auxiliary neurons of all ADD and SUB modules, which is useful in ensuring that the ADD and SUB modules can be reused as many times as necessary to simulate the work of the register machine.

In the ADD and SUB modules, for each neuron σ_l, $l \in H$, a function Lab is defined on the set of instruction labels H:

$$Lab(l) = \begin{cases} l_{i_1}^{(4)}, \text{ for } l_{i_1} : (\texttt{ADD}(r), l, l_k), \\ l_{i_2}^{(5)}, \text{ for } l_{i_2} : (\texttt{ADD}(r), l_j, l), \\ l_{i_3}^{(4)}, \text{ for } l_{i_3} : (\texttt{SUB}(r), l, l_k), \\ l_{i_4}^{(7)}, \text{ for } l_{i_4} : (\texttt{SUB}(r), l_j, l). \end{cases}$$

The function Lab specifies the neurons in different modules from which neuron σ_l requests spikes when an instruction l will be simulated, see Figure 2.

Initially, all neurons in system Π are empty, with the exception of neuron σ_{l_0} containing one spike, which corresponds to the fact that M begins a computation by applying the instruction with label l_0. During a computation, once a neuron σ_{l_i} requests one spike, the module associated with the corresponding instruction $l_i : (\texttt{OP}(r), l_j, l_k)$ of M becomes activated and starts to simulate the instruction: first neuron σ_{l_i} containing one spike, acting on neuron σ_r as requested by OP (OP is the operation of ADD or SUB), and finally neuron σ_{l_j} or σ_{l_k} containing one spike. When neuron σ_{l_h} associated with the halt label l_h requests one spike, it means that the computation in M is completely simulated in Π; at the same time, the computation of Π also halts and the number of spikes in the output neuron σ_1 is the result of the computation. The simulation process is illustrated in Figure 3.

In what follows, we describe the work of the ADD and SUB modules in a graphical form, in order to prove that system Π can correctly simulate

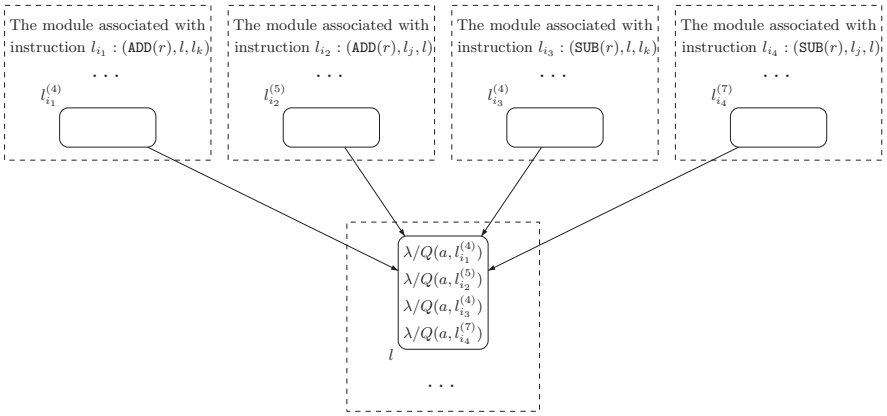

Figure 2. An example of the function *Lab* defined on H in neuron σ_l.

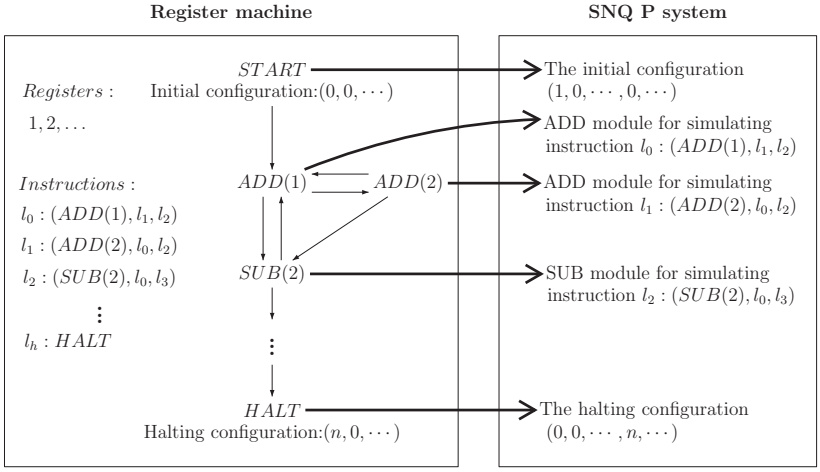

Figure 3. An illustration of the simulation process for an SNQ P system simulating a register machine.

register machine M.

`ADD` module (shown in Figure 4): simulating an ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$.

Assume that at step t, system Π starts to simulate an ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$, and the module associated with the corresponding ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ is activated, which means that neuron σ_{l_i} requests one spike from another neuron. At step $t+1$, with one spike in neuron σ_{l_i}, rules $\lambda/Q(a, l_i)$ in neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, and $\sigma_{l_i^{(3)}}$ are enabled and applied, and each neuron $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, and $\sigma_{l_i^{(3)}}$ requests one spike from neuron σ_{l_i}. At step $t+2$, neuron σ_r requests one spike by using rule $a^*/Q(a, l_i^{(1)})$, which simulates the increase of the number stored in register r by one. Simultaneously, with one spike in neuron $\sigma_{l_i^{(3)}}$, rule $a/Q(a, l_i^{(1)})(a, l_i^{(2)})$ is applied, and two spikes are requested: one each from neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$. In this way, there are three spikes present in neuron σ_{l_i} in total, but the two queries $(a^2, l_i^{(3)})$ of neuron $\sigma_{l_i^{(4)}}$ and $(a^3, l_i^{(3)})$ of neuron $\sigma_{l_i^{(5)}}$ are conflicting, thus one of them is nondeterministically chosen to be satisfied, which leads to one of modules associated with instructions l_j and l_k in system Π nondeterministically activated.

If rule $\lambda/Q(a^2, l_i^{(3)})$ in neuron $\sigma_{l_i^{(4)}}$ is applied at step $t+3$, then two spikes are requested from neuron $\sigma_{l_i^{(3)}}$. In this way, one spike remains in neuron $\sigma_{l_i^{(3)}}$, but no rule inside it can be used, which will be removed by neuron σ_g later. At step $t+4$, with two spikes in neuron $\sigma_{l_i^{(4)}}$, neuron σ_{l_j} requests one spike from neuron $\sigma_{l_i^{(4)}}$ by using rule $\lambda/Q(a, l_i^{(4)})$, and the module associated with instruction l_j is activated, starting to simulate instruction l_j of M. As

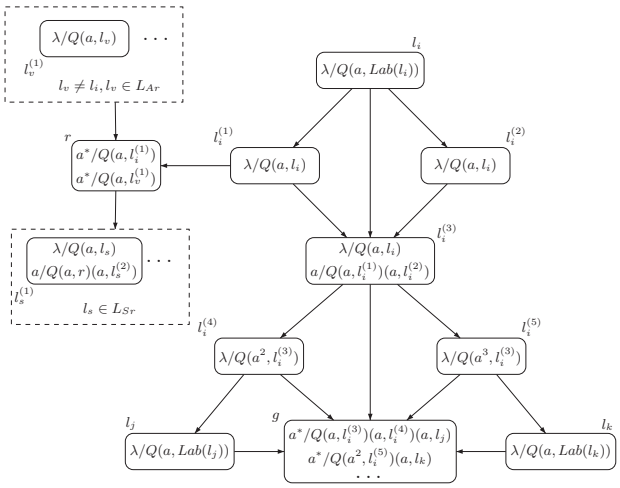

Figure 4. The ADD module of Π simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$, where L_{Ar}=$\{l \mid l$ is a label of an ADD instruction acting on register $r\}$, and L_{Sr}=$\{l \mid l$ is a label of a SUB instruction acting on register $r\}$.

each neuron $\sigma_{l_i^{(3)}}$, $\sigma_{l_i^{(4)}}$, and σ_{l_j} has one spike, rule $a^*/Q(a,l_i^{(3)})(a,l_i^{(4)})(a,l_j)$ in neuron σ_g is enabled and applied, requesting one spike from neurons $\sigma_{l_i^{(3)}}$, $\sigma_{l_i^{(4)}}$, and σ_{l_j}, respectively. Therefore, the numbers of spikes in neurons $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$ are reset to zero as in their initial configuration. The query (a, l_j) in rule $a^*/Q(a,l_i^{(3)})(a,l_i^{(4)})(a,l_j)$ is used to guarantee the application of the rule only in case of the choice of the branch l_j.

If rule $\lambda/Q(a^3,l_i^{(3)})$ in neuron $\sigma_{l_i^{(5)}}$ is applied at step t+3, then neuron $\sigma_{l_i^{(5)}}$ requests three spikes from neuron $\sigma_{l_i^{(3)}}$, and no spike remains in neuron $\sigma_{l_i^{(3)}}$. At step $t+4$, neuron σ_{l_k} requests one spike from neuron $\sigma_{l_i^{(5)}}$ by using rule $\lambda/Q(a,l_i^{(5)})$, and the module associated with instruction l_k is activated, starting to simulate instruction l_k of M. At step $t+5$, the remaining two spikes in neuron $\sigma_{l_i^{(5)}}$ are removed by rule $a^*/Q(a^2,l_i^{(5)})(a,l_k)$ in neuron σ_g, resetting the number of spikes in neuron $\sigma_{l_i^{(5)}}$ to zero. Likewise, the query (a, l_k) in rule $a^*/Q(a^2,l_i^{(5)})(a,l_k)$ is used to guarantee the execution of the rule only in case of the choice of the branch l_k.

An important observation is that in spite of the fact that there are several ADD instructions $l_v \neq l_i$ acting on register r, rules $a^*/Q(a,l_v^{(1)})$ cannot be applied, thus it does not cause undesired steps (i.e., steps in Π that do not correspond to correct simulations of instructions of M). Moreover, all ADD and SUB modules of system Π share a unique "garbage collector" neuron σ_g. However, in each time unit, only one query of neuron σ_g is satisfied. Thus, undesired steps in system Π are not allowed.

During the simulation of ADD instructions, the numbers of spikes in neurons of the ADD module evolve as shown in Tables 1 and 2, which correspond to the two cases in which the modules associated with instructions l_j and l_k are activated, respectively. As we do not care the number of spikes in the "garbage collector" neuron σ_g, the number of spikes inside it is not presented in these two tables. From these two tables, after these steps, all neurons are in their initial configuration with respect to the number of spikes, with the exception of neuron σ_r whose copies of spikes are incremented by one.

Therefore, through the above description about the ADD module, the ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ is correctly simulated: system Π begins with one spike present in neuron σ_{l_i}, and ends with one spike in neuron σ_{l_j} or σ_{l_k}, nondeterministically chosen; meanwhile, the number of spikes in neuron σ_r is increased by one.

SUB module (shown in Figure 5): simulating a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$.

Assume that at step t, a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is simulated in system Π, which means that neuron σ_{l_i} requests one spike from another neuron, and the module associated with SUB instruction l_i is activated. At step $t+1$, with one spike in neuron σ_{l_i}, neurons $\sigma_{l_i^{(1)}}$, $\sigma_{l_i^{(2)}}$, $\sigma_{l_i^{(3)}}$ request one spike from neuron σ_{l_i}, respectively. Neuron $\sigma_{l_i^{(1)}}$ is used to check

Table 1. The evolution of the numbers of spikes in neurons of module ADD during the simulation of ADD instructions with the module associated with instruction l_j finally activated.

Neuron	Step					
	t	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$
σ_{l_i}	1	0	0	0	0	0
σ_r	n	n	$n+1$	$n+1$	$n+1$	$n+1$
$\sigma_{l_i^{(1)}}$	0	1	0	0	0	0
$\sigma_{l_i^{(2)}}$	0	1	0	0	0	0
$\sigma_{l_i^{(3)}}$	0	1	3	1	1	0
$\sigma_{l_i^{(4)}}$	0	0	0	2	1	0
$\sigma_{l_i^{(5)}}$	0	0	0	0	0	0
σ_{l_j}	0	0	0	0	1	0
σ_{l_k}	0	0	0	0	0	0

Table 2. The evolution of the numbers of spikes in neurons of module ADD during the simulation of ADD instructions with the module associated with instruction l_k finally activated.

Neuron	Step					
	t	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$
σ_{l_i}	1	0	0	0	0	0
σ_r	n	n	$n+1$	$n+1$	$n+1$	$n+1$
$\sigma_{l_i^{(1)}}$	0	1	0	0	0	0
$\sigma_{l_i^{(2)}}$	0	1	0	0	0	0
$\sigma_{l_i^{(3)}}$	0	1	3	0	0	0
$\sigma_{l_i^{(4)}}$	0	0	0	0	0	0
$\sigma_{l_i^{(5)}}$	0	0	0	3	2	0
σ_{l_j}	0	0	0	0	0	0
σ_{l_k}	0	0	0	0	1	0

whether the number of spikes in neuron σ_r is zero. For neuron $\sigma_{l_i^{(1)}}$, there are the following two cases:

(1) At step t, if there is no spike present in neuron σ_r, which corresponds to the fact that the number stored in register r is zero, then rule $a/Q(a,r)(a,l_i^{(2)})$ in neuron $\sigma_{l_i^{(1)}}$ cannot be used at step $t+2$, and there is one spike remaining in each neuron $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$. At step $t+2$, neuron $\sigma_{l_i^{(5)}}$ requests one spike from neuron $\sigma_{l_i^{(3)}}$. As neuron $\sigma_{l_i^{(2)}}$ has one spike, at step $t+3$, rule $\lambda/Q(a,l_i^{(5)})(a,l_i^{(2)})$ in neuron $\sigma_{l_i^{(6)}}$ is enabled and applied, and neuron $\sigma_{l_i^{(6)}}$ requests two spikes each from neurons $\sigma_{l_i^{(2)}}$ and $\sigma_{l_i^{(5)}}$. At step $t+4$, with one spike in neuron $\sigma_{l_i^{(1)}}$ and two spikes in $\sigma_{l_i^{(6)}}$, rule $\lambda/Q(a^2,l_i^{(6)})(a,l_i^{(1)})$ is applied, and neuron $\sigma_{l_i^{(7)}}$ requests three spikes in total. In this way, neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$ now have no spike, so the execution of the branch l_j is stopped and the whole branch is reset to its original state in terms of the number of spikes. At step $t+5$, neuron σ_{l_k} requests one spike from neuron $\sigma_{l_i^{(7)}}$, and the module associated with instruction l_k is activated, starting to simulate instruction l_k of M. The remaining two spikes in neuron $\sigma_{l_i^{(7)}}$ are removed by rule $a^*/Q(a^2,l_i^{(7)})(a,l_k)$ in neuron σ_g at step $t+6$.

(2) At step t, if there is at least one spike present in neuron σ_r, which corresponds to the fact that the number stored in register r is larger than zero, then rule $a/Q(a,r)(a,l_i^{(2)})$ in neuron $\sigma_{l_i^{(1)}}$ is applied at step $t+2$, one spike is requested from neuron σ_r, which simulates that the number stored in register r is decreased by one, as well as one spike is requested from neuron $\sigma_{l_i^{(2)}}$. In this way, neuron $\sigma_{l_i^{(1)}}$ accumulates three spikes. At step $t+3$, as neuron $\sigma_{l_i^{(2)}}$ has no spike,

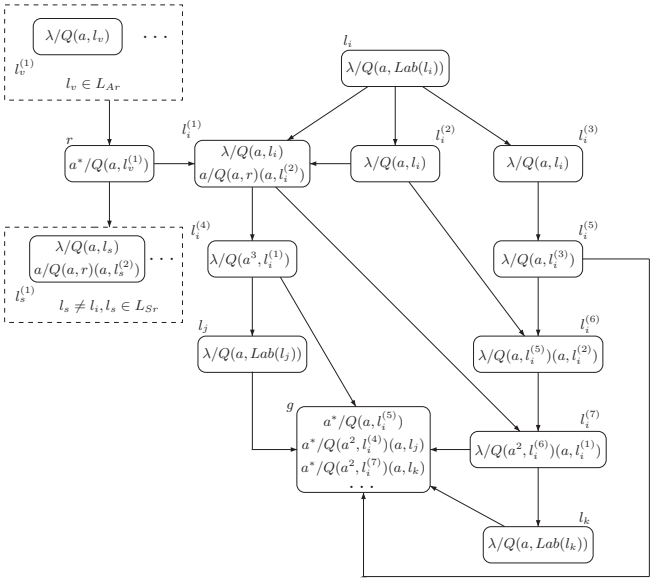

Figure 5. The SUB module of Π simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$, where L_{Ar}=$\{l \mid l$ is a label of an ADD instruction acting on register $r\}$, and $L_{Sr} = \{l \mid l$ is a label of a SUB instruction acting on register $r\}$.

rule $\lambda/Q(a, l_i^{(5)})(a, l_i^{(2)})$ in neuron $\sigma_{l_i^{(6)}}$ cannot be used, resulting in one spike remaining in neuron $\sigma_{l_i^{(5)}}$. Yet the spike is removed by rule $a^*/Q(a^2, l_i^{(5)})$ in neuron σ_g at this step. Simultaneously, neuron $\sigma_{l_i^{(4)}}$ requests three spikes from neuron $\sigma_{l_i^{(1)}}$ by using rule $\lambda/Q(a^3, l_i^{(1)})$. In the next step, neuron σ_{l_j} requests one spike from neuron $\sigma_{l_i^{(4)}}$, and the module associated with instruction l_j is activated, starting to simulate instruction l_j of M, while the remaining two spikes in neuron $\sigma_{l_i^{(4)}}$ are removed by rule $a^*/Q(a^2, l_i^{(4)})(a, l_j)$ in neuron σ_g at step t+5. Note that the rules in all neurons $\sigma_{l_i^{(6)}}$, $\sigma_{l_i^{(7)}}$, and σ_{l_k} cannot be used, so the execution of the branch l_k cannot be activated.

During the simulation of SUB instructions, the numbers of spikes in neurons of the SUB module evolve as presented in Tables 3 and 4, which correspond to the the two cases in which, at step t, the number stored in register r is zero and the number stored in register r is larger than zero, respectively. Similarly, the number of spikes inside neuron σ_g is not presented in these two tables.

Note that even if there are multiple SUB instructions $l_s \neq l_i$ acting on register r, rule $a/Q(a, r)(a, l_s^{(2)})$ in neuron $\sigma_{l_s^{(1)}}$ cannot be applied, thus no undesired steps can happen. Similarly, if there are multiple ADD instructions $l_v \neq l_i$ acting on register r, since neuron $\sigma_{l_v^{(1)}}$ has no spike, neuron σ_r cannot request spikes from it. In this way undesired steps will be avoided.

From the above explanation, the simulation of the SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is correct: system Π starts with one spike contained in neuron σ_{l_i}, and ends with one spike in neuron σ_{l_k} (if the number stored in register r is zero), or one spike in neuron σ_{l_j} with the number of spikes in neuron σ_r decreased by one (if the number stored in register r is larger than zero).

Assume that at step t neuron σ_{l_h} requests one spike from another neuron, which means that the computation in M halts, i.e., the halt instruction is reached. Even if neuron σ_{l_h} has one spike, there is no other neuron requesting spikes from it. In this way, the computation in system Π halts, and the number of spikes in neuron σ_1 is the result of the computation, which is exactly the number stored in register 1 when M halts.

Based on the aforementioned description of the work about the ADD and SUB modules, register machine M is correctly simulated by system Π, i.e., $N_{all}(\Pi) = N(M)$. We can check that only one type of spike is used in system Π, the number of neu-

Table 3. The evolution of the numbers of spikes in neurons of module SUB during the simulation of SUB instructions in the case that the number of spikes in neuron σ_r is zero.

Neuron	Step						
	t	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$	$t+6$
σ_{l_i}	1	0	0	0	0	0	0
σ_r	0	0	0	0	0	0	0
$\sigma_{l_i^{(1)}}$	0	1	1	1	0	0	0
$\sigma_{l_i^{(2)}}$	0	1	1	0	0	0	0
$\sigma_{l_i^{(3)}}$	0	1	0	0	0	0	0
$\sigma_{l_i^{(4)}}$	0	0	0	0	0	0	0
$\sigma_{l_i^{(5)}}$	0	0	1	0	0	0	0
$\sigma_{l_i^{(6)}}$	0	0	0	2	0	0	0
$\sigma_{l_i^{(7)}}$	0	0	0	0	3	2	0
σ_{l_j}	0	0	0	0	0	0	0
σ_{l_k}	0	0	0	0	0	1	0

Table 4. The evolution of the numbers of spikes in neurons of module SUB during the simulation of SUB instructions in the case that the number of spikes in neuron σ_r is larger than zero.

Neuron	Step					
	t	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$
σ_{l_i}	1	0	0	0	0	0
σ_r	n	n	$n-1$	$n-1$	$n-1$	$n-1$
$\sigma_{l_i^{(1)}}$	0	1	3	0	0	0
$\sigma_{l_i^{(2)}}$	0	1	0	0	0	0
$\sigma_{l_i^{(3)}}$	0	1	0	0	0	0
$\sigma_{l_i^{(4)}}$	0	0	0	3	2	0
$\sigma_{l_i^{(5)}}$	0	0	1	0	0	0
$\sigma_{l_i^{(6)}}$	0	0	0	0	0	0
$\sigma_{l_i^{(7)}}$	0	0	0	0	0	0
σ_{l_j}	0	0	0	0	1	0
σ_{l_k}	0	0	0	0	0	0

rons is not bounded, and the number of unbounded neurons depends on the number of registers in the simulated register machine, so the number of unbounded neurons is also not bounded. Consequently, $N_{all}SN_1QP_*(unb_*) = NRE$, and this concludes the proof. □

In what follows, we construct a specific computing universal SNQ P system with one type of spike by simulating the universal register machine used in Ref. 44. This universal register machine consists of 9 registers, 25 instruction labels, 11 ADD instructions, and 13 SUB instructions.

In the computing universal SNQ P system, the input and the result of a computation are introduced in specified neurons as defined in the same way as in register machines, i.e., the numbers are encoded with the multiplicity of spikes, thus the input and output modules are not needed. In this way, based on the ADD and SUB modules constructed in the proof of Theorem 1, the total number of neurons in the universal SNQ P system with one type of spike can be calculated as follows:

- 9 neurons for the 9 registers,
- 25 neurons for the 25 instruction labels,
- 11 × 5 neurons for the 11 ADD instructions,
- 13 ×7 neurons for the 13 SUB instructions,
- one unique garbage neuron σ_g,

which, in total, is 181 neurons. Therefore, the following corollary is formulated.

Corollary 1. *There exists a computing universal SNQ P system with one type of spike consisting of 181 neurons.*

In Theorem 1, the number of unbounded neurons is not bounded, as it depends on the number of registers in the simulated register machine. It is known that nondeterministic register machines with three registers can still generate all sets of Turing computable natural numbers, hence they characterize NRE.[71] Consequently, the following corollary is obtained.

Corollary 2. *SNQ P systems functioning as number generating devices with four unbounded neurons are Turing universal:*

$$N_{all}SN_1QP_*(unb_4) = NRE$$

Proof. Let us consider a register machine $M' = \{3, H, l_0, l_m, I\}$ with three registers, where the three registers are labeled by 1, 2, and 3, respectively, and the other properties are the same as register machine M specified in the proof of Theorem 1.

Analogous to the proof Theorem 1, an SNQ P system Π' is constructed to simulate M', which consists of two types of modules: the ADD and SUB modules. The structures and the functioning of modules ADD and SUB remain the same as the ones constructed in the proof Theorem 1, as shown in Figures 4 and 5, respectively.

As register machine M' has only three registers, there are four unbounded neurons in system Π', i.e., neurons σ_1, σ_2, and σ_3, which are associated with registers 1, 2, and 3 of M', and a distinguished neuron σ_g, which is used to remove the redundant spikes in some auxiliary neurons of all ADD and SUB modules. Therefore, the corollary holds. □

In Corollary 2, an upper bound on the number of unbounded neurons is obtained for generative SNQ P systems achieving Turing universality.

4. Universality of SNQ P systems with one type of spike working as number acceptors

In this section, the computation power of SNQ P systems working as number acceptors is investigated, and it is proved that SNQ P systems working as number acceptors are Turing universal.

When an SNQ P system works as a number acceptor, an input neuron σ_{in} is designed to receive spikes from the environment. A number n is introduced into the system through the input neuron by reading a spiking train

$$1\underbrace{0\ldots0}_{n-1}1$$

(also denoted by $10^{n-1}1$), where the occurrence of digit 1 (resp., 0) indicates that the input neuron receives a spike (resp., no spike). If the computation of the system halts, then this number is said to be accepted.

Theorem 2. *SNQ P systems with one type of spike in the accepting mode are Turing universal:*

$$N_{acc}SN_1QP_*(unb_*) = NRE. \quad (1)$$

Proof. This proof is based on simulating deterministic register machines working in the accepting mode. It is known that deterministic register machines can also characterize NRE. We construct a specific SNQ P system Π_d to simulate a deterministic register machine $M_d = (m, H, l_0, l_h, I)$ working in the accepting mode. The definition of deterministic register machines is similar to nondeterministic register machines, with the exception that the ADD instructions are of the form $l_i : (\texttt{ADD}(r), l_j)$ (add 1 to register r, then precisely go to the instruction with label l_j). The set of all the numbers accepted by deterministic register machine M_d is denoted by $N_{acc}(M_d)$.

System Π_d consists of an INPUT module, the deterministic ADD and SUB modules. The deterministic ADD and SUB modules are used to simulate the deterministic ADD and SUB instructions of M_d, respectively; the INPUT module is designed to load the necessary spikes into a specified neuron and activate the simulation of a computation in M_d.

Analogously, each register r of M_d corresponds to a neuron σ_r in system Π_d, and the number of spikes in neuron σ_r is equal to the number stored in register r. Each label l_i of an instruction in M_d also corresponds to a neuron σ_{l_i} and some auxiliary neurons $\sigma_{l_i^{(j)}}$, $j = 1, 2, \ldots$, in system Π_d. There is also a distinguished neuron σ_g in system Π_d to remove the remaining spikes in some auxiliary neurons of the INPUT and all SUB modules.

The INPUT module is presented in Figure 6, which is used to load n spikes into neuron σ_1 through neuron σ_{in} by reading a spike train $10^{n-1}1$ from the environment, which corresponds to number n analyzed by system Π_d. If the computation of system Π_d halts, then number n is said to be accepted. The

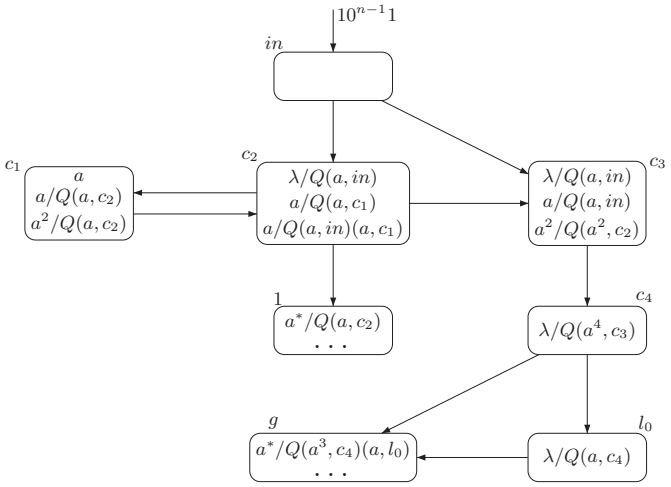

Figure 6. The INPUT module of Π_d.

input neuron is identified by label in, and pictorially by an incoming arrow entering the neuron, which suggests that it communicates with the environment.

The INPUT module works in the following way. Initially, all neurons are empty with the exception that neuron σ_{c_1} contains one spike. Assume that at step t the first spike arrives in neuron σ_{in} from the environment. At step $t+1$, each neuron σ_{c_2} and σ_{c_3} requests one spike from neuron σ_{in} by using rules $\lambda/Q(a,in)$. In the next step, with one spike inside, rules $a/Q(a,c_2)$ and $a/Q(a,c_1)$ in neurons σ_{c_1} and σ_{c_2} are applied, requesting one spike from each other. Actually, from step $t+2$ on, neurons σ_{c_1} and σ_{c_2} requests one spike from each other in each step. Moreover, from step $t+2$ on, rule $a^*/Q(a,c_2)$ in neuron σ_1 is applied in each step, requesting one spike from neuron σ_{c_2}, which simulates that the number stored in register 1 is increased by one, until neuron σ_{c_2} requests the second spike from neuron σ_{in}.

At step $t+n$, the second spike arrives in neuron σ_{in} from the environment. At step $t+n+1$, neuron σ_{c_3} requests the second spike from neuron σ_{in} by using rule $a/Q(a,in)$. In this way, neuron σ_{c_3} accumulates two spikes. Simultaneously, neurons σ_1 and σ_{c_1} request one spike from neuron σ_{c_2}, respectively. For neuron σ_{c_2}, rules $a/Q(a,in)(a,c_1)$ and $a/Q(a,c_1)$ are enabled, and one of them is nondeterministically chosen to be applied.

Case I. If rule $a/Q(a,in)(a,c_1)$ is applied at step $t+n+1$, then neuron σ_{c_2} requests two spikes each from neurons σ_{in} and σ_{c_1}. In this case, neuron σ_{c_2} accumulates two spikes. In the next step, however, the two queries (a,c_2) of neurons σ_1 and σ_{c_1} and (a^2,c_2) of neuron σ_{c_3} are conflicting, and one of them is nondeterministically chosen to be satisfied. Thus, there are also the following two cases.

(1) If rule $a^2/Q(a^2,c_2)$ in neuron σ_{c_3} is applied at step $t+n+2$, then neuron σ_{c_3} requests two spikes from neuron σ_{c_2}. In this case, neuron σ_{c_2} has no spike, so neurons σ_1 and σ_{c_1} cannot request spikes from neuron σ_{c_2} any more. From step $t+2$ to step $t+n+1$, neuron σ_1 accumulates n spikes in total, which is just the number to be analyzed. At step $t+n+3$, neuron σ_{c_4} requests four spikes from neuron σ_{c_3} by using rule $\lambda/Q(a^4,c_3)$. One step later, neuron σ_{l_0} requests one spike from neuron σ_{c_4}, which means that the module associated with the instruction with label l_0 of M_d is activated; the remaining three spikes in neuron σ_{c_4} are removed by using rule $a^*/Q(a^3,c_4)(a,l_0)$ in "garbage collector" neuron σ_g at step $t+n+5$.

(2) If rules $a^*/Q(a,c_2)$ and $a/Q(a,c_2)$ in neurons σ_1 and σ_{c_1} are applied at step $t+n+2$, then neurons σ_1 and σ_{c_1} request one spike from neuron σ_{c_2}, respectively. In this case, one spike remains in neuron σ_{c_2}, so the query (a^2,c_2) of neuron σ_{c_3} is not satisfied, and no rule inside it can be applied. In this way, the module associated with the instruction with label l_0 cannot be activated forever. From

step $t+n+3$ on, neurons c_1 and c_2 will repeatedly request one spike from each other by using rules $a^2/Q(a,c_2)$ and $a/Q(a,c_1)$, and neuron σ_1 also continuously requests one spike from neuron σ_{c_2}, thus the computation of system Π_d never stops.

Case II. If rule $a/Q(a,c_1)$ in neuron σ_{c_2} is applied at step $t+n+1$, then neuron σ_{c_2} contains only one spike. As the second spike in neuron σ_{in} is requested by neuron σ_{c_3} at this step, neuron σ_{c_2} can only use rule $a/Q(a,c_1)$. Thus, neurons c_1 and c_2 will continuously use rules $a/Q(a,c_2)$ and $a/Q(a,c_1)$ to request one spike from each other, and the computation of system Π_d also never halts. This case is similar to the case of item (2) of Case I.

The numbers of spikes in neurons of the INPUT module evolve during the process initializing the computation as shown in Table 5, corresponding to the case in which the initialization process is correctly completed.

In system Π_d, the function Lab' defined on the set of instruction labels H in each neuron σ_l, $l \in H$ is presented as follows:

$$Lab'(l) = \begin{cases} l_{i_1}^{(1)}, \text{ for } l_{i_1} : (\texttt{ADD}(r), l), \\ l_{i_2}^{(4)}, \text{ for } l_{i_2} : (\texttt{SUB}(r), l, l_k), \\ l_{i_3}^{(7)}, \text{ for } l_{i_3} : (\texttt{SUB}(r), l_j, l). \end{cases}$$

Figure 7 displays the proposed deterministic ADD module corresponding to the instruction $l_i : (\texttt{ADD}(r), l_j)$. It can be observed from the scheme in Figure 7 that the module has been greatly simplified when compared with its original counterpart shown in Figure 4. Its functioning is rather clear, we here omit the description of the work of the deterministic ADD module.

The SUB module remains unchanged as the one constructed in the proof of Theorem 1, shown in Figure 5. The simulation of the computation in M_d proceeds until neuron σ_{l_h} requests one spike, which means that the halt instruction is reached and the computation in M_d halts. Although neuron σ_{l_h} has one spike, no other neuron requests the spike from it. In this way, the computation of system Π_d halts.

From the aforementioned description of the work of system Π_d, deterministic register machine M_d is correctly simulated, i.e., $N_{acc}(\Pi_d) = N_{acc}(M_d)$. It is easy to check that only one type of spike is used in system Π_d, the number of neurons and the number of unbounded neurons are not bounded. Therefore, this implicates that $N_{acc}SN_1QP_*(unb_*) = NRE$. □

5. Conclusions

In this work, we investigated the computation power of SNQ P systems with one type of spike. It was proved that one type of spike suffices for SNQ P systems achieving Turing universality as both number generating and number accepting devices. This result answers the corresponding open problem formulated in Ref. 61 and contains interesting practical implications: SNQ P systems, even when dramatically simplified are still equivalent to Turing machines. The reduction to only one type of spike leads to computationally cheaper integration-and-fire conditions (since the alphabet O is composed of only one letter) without a deterioration of the computation power.

We also investigated the influence of the number of unbounded neurons on the computation power of SNQ P systems. It was proved that SNQ P systems with four unbounded neurons are Turing universal as number generating devices. This result, albeit a corollary of the universality of SNQ P systems with one type of spike, is also a preliminary result for future research. The number of unbounded neurons will be interpreted as a parameter to tune and control the Turing universality of SNQ P systems. Further research will investigate the relationship between the number of unbounded neurons and the complexity of an SNQ P system.

5.1. *Comparisons with Other Computation Models*

In Corollary 1, we have constructed a computing universal SNQ P system with one type of spike. In order to validate the performance on achieving Turing universality of SNQ P systems with one type of spike, in terms of lower bound on the number of neurons, we compare them with SNQ P systems working in the local check mode with one type of spike, SNQ P systems with two types of spikes, SN P systems, and recurrent neural networks. The numbers of neurons for achieving Turing universality of five computation models are shown in Table 6.

Compared with the universal SNQ P system

Table 5. The evolution of the numbers of spikes in neurons of the INPUT module during the process initializing the computation in the case that the initialization process correctly completed.

Neuron	Step										
	t	t+1	t+2	t+3	$\cdots$	t+n	t+n+1	t+n+2	t+n+3	t+n+4	t+n+5
σ_{in}	1	0	0	0	$\cdots$	1	0	0	0	0	0
σ_1	0	0	1	2	$\cdots$	$n-1$	n	n	n	n	n
σ_{c_1}	1	1	1	1	$\cdots$	1	1	1	1	1	1
σ_{c_2}	0	1	1	1	$\cdots$	1	2	0	0	0	0
σ_{c_3}	0	1	1	1	$\cdots$	1	2	4	0	0	0
σ_{c_4}	0	0	0	0	$\cdots$	0	0	0	4	3	0
σ_{l_0}	0	0	0	0	$\cdots$	0	0	0	0	1	0

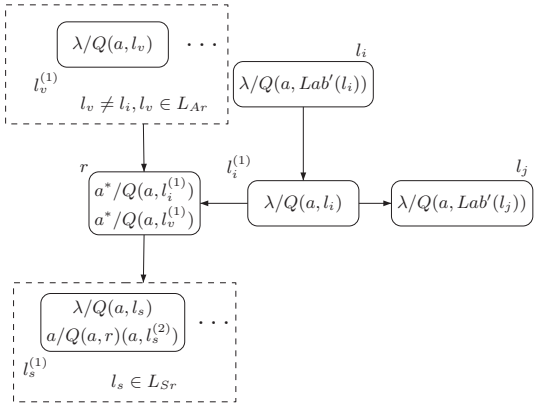

Figure 7. The deterministic ADD module of Π_d simulating $l_i : (\texttt{ADD}(r), l_j)$, where L_{Ar}=$\{l \mid$ is a label of an ADD instruction acting on register $r\}$, and L_{Sr}=$\{l \mid$ is a label of a SUB instruction acting on register $r\}$.

with one type of spike constructed in Corollary 1, the number of neurons needed for constructing the universal SNQ P system working in the local check mode with one type of spike is significantly decreased,[73] specifically reducing from 181 neurons to 63 neurons. This result implies that the semantics of rule application can affect the computation capability of SNQ P systems.

In Ref. 61, a universal SNQ P system with two types of spikes for computing functions was constructed by using 49 neurons. By comparing with the two universal SNQ P systems with one type of spike, the number of neurons in the universal SNQ P system with two types of spikes is smaller, but the number of types of spikes becomes larger. From biological point of view, as spikes have an almost identical shape (voltage, duration, etc.) and the form of spikes does not carry any information, considering multiple types of spikes in SNQ P systems is rather non-realistic, thus it is more realistic to use one type of spike to encode information in SNQ P systems. SNQ P systems with one type of spike are also consistent with the definition of classical SN P systems. Furthermore, from the practical point of view, SNQ P systems with one type of spike are more feasible to be constructed, particularly for actualizing neural-like computation models by using electronic materials.

SN P systems are a class of spiking neural networks in the framework of membrane computing, where neurons evolve by applying spiking and forgetting rules in the form of formal grammar production, instead of applying mathematically-defined functions in artificial neural networks. A universal SN P system for computing functions was constructed by using 84 neurons.[44] By Table 6, the number of neurons for SN P system achieving universality is smaller than that of the universal SNQ P system constructed in Corollary 1. These results imply that the communication patterns have an influence on the computation capability of SNQ P systems.

Recurrent neural networks are a class of well-known artificial neural networks. A recurrent neural network consists of interconnections of neurons,

Table 6. Number of neurons for achieving universality of five computation models.

Computation Models	Number of Neurons
SNQ P systems with one type of spike	181
SNQ P systems working in the local check mode with one type of spike[73]	63
SNQ P systems with two types of spikes[61]	49
SN P systems[44]	84
Recurrent neural networks[74]	886

where all neurons synchronously evolve, each updating its state and producing a real-valued activation by applying a sigmoidal function to combine the previous states of all afferent neurons. It was shown that 886 neurons are needed for a recurrent neural network to achieve Turing universality for computing functions.[74] From Table 6, the numbers of neurons for achieving universality in the four other computation models are much smaller than that of recurrent neural network. These results reveal the integrate-and-fire condition in the form of regular expressions for SNQ P systems achieving a powerful computation capability.

5.2. *Real-world applicability of simplified SNQ P Systems*

In SNNs, the timing of individual spikes is used to encode information. The temporal dimension for information encoding in SNNs yields to a great potential for solving time-dependent tasks, e.g., temporal pattern recognition, time-series forecasting. These time-dependent application domains have an additional dimension of time, and the temporal dimension provides a wealth of additional information, thus not just the immediate data is important, but the sequence of data. SNNs have shown the capability of dealing with various application domains by incorporating temporal dynamics into information processing. Moreover, the temporal information encoding has a powerful capacity in terms of the wide range of information that is encoded by the same number of neurons; thus it can reduce the number of neurons that are necessary to perform a given task.

Since SNQ P systems are a type of SNNs, they also share with them the temporal information encoding. In this work, we have shown that even simplified SNQ P systems still have the powerful computation capability. Thus, we believe that the proposed SNQ P systems with a single type of spikes are a promising computational tool in some scenarios. For instance, SNQ P systems can be used in the field of pattern recognition, such as the recognition of characters. SNNs have been used to recognize characters, e.g., in Ref. 75, and most of characters can be recognized uniquely in the sense that either a unique neuron fires or the firing times of the same neuron are different. However, SNNs cannot achieve a good performance for recognizing the characters with the same number of pixels. All of the characters M, U, and W have 11 pixels, although they are represented by a different array of 3×5 pixels, and SNNs cannot well recognize the characters M, U, and W. SNQ P systems have the potential to overcome the difficulty. Multiple rules with different spiking conditions enable a neuron of an SNQ P system to work as an intelligent "selector" choosing desired spiking pattern when the neuron reads different arrays with the same number of pixels. That is, SNQ P systems have the ability to distinguish different arrays with the same number of pixels, and thus recognizing the characters with the same number of pixels. Of course, the real performance of SNQ P systems in recognizing characters needs further experiments to be verified.

Acknowledgments

This work was supported by National Natural Science Foundation of China (61320106005 and 61772214), the Innovation Scientists and Technicians

Troop Construction Projects of Henan Province (154200510012), and ANCS under award number POC P 37 257 (MoDASyS).

Bibliography

1. Y. Zeinali and B. A. Story, Competitive probabilistic neural network, *Integrated Computer-Aided Engineering* **24**(2) (2017) 105–118.
2. A. Rigos, G. E. Tsekouras, M. I. Vousdoukas, A. Chatzipavlis and A. F. Velegrakis, A chebyshev polynomial radial basis function neural network for automated shoreline extraction from coastal imagery, *Integrated Computer-Aided Engineering* **23**(2) (2016) 141–160.
3. J. A. Garrido, N. R. Luque, S. Tolu and E. DAngelo, Oscillation-driven spike-timing dependent plasticity allows multiple overlapping pattern recognition in inhibitory interneuron networks, *International Journal of Neural Systems* **26**(05) (2016) p. 1650020.
4. S. Iliya and F. Neri, Towards artificial speech therapy: A neural system for impaired speech segmentation, *International Journal of Neural Systems* **26**(6) (2016) p. 1650023.
5. T. J. Hirschauer, H. Adeli and J. A. Buford, Computer-aided diagnosis of parkinson's disease using enhanced probabilistic neural network, *J. Medical Systems* **39**(11) (2015) p. 179.
6. S. Schliebs, N. Kasabov and M. Defoin-Platel, On the probabilistic optimization of spiking neural networks, *International Journal of Neural Systems* **20**(06) (2010) 481–500.
7. S. Shapero, M. Zhu, J. Hasler and C. Rozell, Optimal sparse approximation with integrate and fire neurons, *International Journal of Neural Systems* **24**(05) (2014) p. 1440001.
8. D. R. Freestone, K. J. Layton, L. Kuhlmann and M. J. Cook, Statistical performance analysis of data-driven neural models, *International Journal of Neural Systems* **27**(1) (2017) 1–16.
9. D. Gentiletti, P. Suffczynski, V. Gnatkovsky and M. de Curtis, Changes of ionic concentrations during seizure transitions - A modeling study, *International Journal of Neural Systems* **27**(4) (2017) 1–16.
10. L. Appeltant, S. M. C, G. V. der Sande, J. Danckaert, S. Massar, J. Dambre, C. R. Mirasso and I. Fischer, Information processing using a single dynamical node as complex system, *Nature Communications 2* (468) (2011).
11. L. Pigou, S. Dieleman, P.-J. Kindermans and B. Schrauwen, *Sign Language Recognition Using Convolutional Neural Networks*, *Computer Vision - ECCV 2014 Workshops: Zurich, Switzerland, September 6-7 and 12, 2014, Proceedings, Part I* eds. L. Agapito, M. M. Bronstein and C. Rother (Springer International Publishing, 2015), pp. 572–578.
12. H. T. Huynh, Y. Won and J.-j. Kim, An improvement of extreme learning machine for compact single-hidden-layer feedforward neural networks, *International Journal of Neural Systems* **18**(05) (2008) 433–441.
13. M. Ahmadlou and H. Adeli, Enhanced probabilistic neural network with local decision circles: A robust classifier, *Integrated Computer-Aided Engineering* **17**(3) (2010) 197–210.
14. N. Wang and H. Adeli, Self-constructing wavelet neural network algorithm for nonlinear control of large structures, *Eng. Appl. of AI* **41** (2015) 249–258.
15. W. Maass, Networks of spiking neurons: the third generation of neural network models, *Neural Networks* **10**(9) (1997) 1659–1671.
16. S. Ghosh-Dastidar and H. Adeli, Spiking neural networks, *International Journal of Neural Systems* **19**(04) (2009) 295–308.
17. S. M. Bohte, J. N. Kok and H. La Poutre, Error-backpropagation in temporally encoded networks of spiking neurons, *Neurocomputing* **48**(1) (2002) 17–37.
18. F. Ponulak and A. Kasiński, Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting, *Neural Computation* **22**(2) (2010) 467–510.
19. O. Booij and H. Tat Nguyen, A gradient descent rule for spiking neurons emitting multiple spikes, *Information Processing Letters* **95**(6) (2005) 552–558.
20. T. J. Strain, L. McDaid, T. M. McGinnity, L. P. Maguire and H. M. Sayers, An STDP training algorithm for a spiking neural network with dynamic threshold neurons, *International Journal of Neural Systems* **20**(06) (2010) 463–480.
21. F. Montani, A. Oliynyk and L. Fadiga, Superlinear summation of information in premotor neuron pairs, *Int. J. Neural Syst.* **27**(2) (2017) 1–24.
22. S. Ghosh-Dastidar and H. Adeli, Improved spiking neural networks for EEG classification and epilepsy and seizure detection, *Integrated Computer-Aided Engineering* **14**(3) (2007) 187–212.
23. S. Ghosh-Dastidar and H. Adeli, A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection, *Neural networks* **22**(10) (2009) 1419–1431.
24. A. Mohemmed, S. Schliebs, S. Matsuda and N. Kasabov, Span: Spike pattern association neuron for learning spatio-temporal spike patterns, *International Journal of Neural Systems* **22**(04) (2012) p. 1250012.
25. E. Nichols, L. McDaid and N. Siddique, Case study on a self-organizing spiking neural network for robot navigation, *International Journal of Neural Systems* **20**(06) (2010) 501–508.
26. A. Geminiani, C. Casellato, A. Antonietti, E. DAngelo and A. Pedrocchi, A multiple-plasticity spiking neural network embedded in a closed-loop control system to model cerebellar pathologies, *Interna-*

tional Journal of Neural Systems (2017) p. 1750017, to appear.
27. C. Luo, Y. Zhang, W. Cao, Y. Huang, F. Yang, J. Wang, S. Tu, X. Wang and D. Yao, Altered structural and functional feature of striato-cortical circuit in benign epilepsy with centrotemporal spikes, *International Journal of Neural Systems* **25**(6) (2015) p. 1550027.
28. L. Guo, Z. Wang, M. Cabrerizo and M. Adjouadi, A cross-correlated delay shift supervised learning method for spiking neurons with application to interictal spike detection in epilepsy, *International Journal of Neural Systems* **27**(3) (2017) p. 1750002.
29. S. Knieling, K. S. Sridharan, P. Belardinelli, G. Naros, D. Weiss, F. Mormann and A. Gharabaghi, An unsupervised online spike-sorting framework, *International Journal of Neural Systems* **26**(5) (2016) p. 1550042.
30. J. A. Garrido, N. R. Luque, S. Tolu and E. D'Angelo, Oscillation-driven spike-timing dependent plasticity allows multiple overlapping pattern recognition in inhibitory interneuron networks, *International Journal of Neural Systems* **26**(5) (2016) p. 1650020.
31. A. Morro, V. Canals, A. Oliver, M. Alomar, F. Galán-Prado, P. Ballester and J. Rosselló, A stochastic spiking neural network for virtual screening, *IEEE Transactions on Neural Networks and Learning Systems* (2017).
32. J. L. Rossello, V. Canals, A. Oliver and A. Morro, Studying the role of synchronized and chaotic spiking neural ensembles in neural information processing, *International Journal of Neural Systems* **24**(05) (2014) p. 1430003.
33. J. L. Rosselló, M. L. Alomar, A. Morro, A. Oliver and V. Canals, High-density liquid-state machine circuitry for time-series forecasting, *International Journal of Neural Systems* **26**(5) (2016) p. 1550036.
34. M. Ionescu, G. Păun and T. Yokomori, Spiking neural P systems, *Fundamenta Informaticae* **71**(2, 3) (2006) 279–308.
35. G. Zhang, H. Rong, F. Neri and M. J. Pérez-Jiménez, An optimization spiking neural P system for approximately solving combinatorial optimization problems, *International Journal of Neural Systems* **24**(05) (2014) p. 1440006.
36. G. Păun, Computing with membranes, *Journal of Computer and System Sciences* **61**(1) (2000) 108–143.
37. G. Păun, *Membrane Computing: An Introduction* (Springer-Verlag, Berlin, 2012).
38. T. Song, L. Pan, J. Wang, I. Venkat, K. G. Subramanian and R. Abdullah, Normal forms of spiking neural p systems with anti-spikes, *IEEE Transactions on NanoBioscience* **11** (Dec 2012) 352–359.
39. M. Cavaliere, O. H. Ibarra, G. Păun, O. Egecioglu, M. Ionescu and S. Woodworth, Asynchronous spiking neural P systems, *Theoretical Computer Science* **410**(24) (2009) 2352–2364.
40. T. Song, L. Pan and G. Păun, Asynchronous spiking neural P systems with local synchronization, *Information Sciences* **219** (2013) 197–207.
41. G. Păun, M. J. Pérez-Jiménez and G. Rozenberg, Spike trains in spiking neural P systems, *International Journal of Foundations of Computer Science* **17**(04) (2006) 975–1002.
42. Z. Xu, M. Cavaliere, P. An, S. Vrudhula and Y. Cao, The stochastic loss of spikes in spiking neural p systems: Design and implementation of reliable arithmetic circuits, *Fundamenta Informaticae* **134** (2014).
43. M. Cavaliere and I. Mura, Experiments on the reliability of stochastic spiking neural p systems, *Natural Computing* **7** (Dec 2008) 453–470.
44. A. Păun and G. Păun, Small universal spiking neural P systems, *BioSystems* **90**(1) (2007) 48–60.
45. H. Chen, R. Freund, M. Ionescu, G. Păun and M. J. Pérez-Jiménez, On string languages generated by spiking neural P systems, *Fundamenta Informaticae* **75**(1–4) (2007) 141–162.
46. T.-O. Ishdorj and A. Leporati, Uniform solutions to SAT and 3-SAT by spiking neural P systems with pre-computed resources, *Natural Computing* **7**(4) (2008) 519–534.
47. T.-O. Ishdorj, A. Leporati, L. Pan, X. Zeng and X. Zhang, Deterministic solutions to QSAT and Q3SAT by spiking neural P systems with pre-computed resources, *Theoretical Computer Science* **411**(25) (2010) 2345–2358.
48. L. Pan, G. Păun and M. J. Pérez-Jiménez, Spiking neural P systems with neuron division and budding, *Science China Information Sciences* **54**(8) (2011) 1596–1607.
49. D. Díaz-Pernil, F. Peña-Cantillana and M. A. Gutiérrez-Naranjo, A parallel algorithm for skeletonizing images by using spiking neural P systems, *Neurocomputing* **115** (2013) 81–91.
50. H. Peng, J. Wang, M. J. Pérez-Jiménez, H. Wang, J. Shao and T. Wang, Fuzzy reasoning spiking neural P systems for fault diagnosis, *Information Sciences* **235** (2013) 106–116.
51. T. Wang, G. Zhang, J. Zhao, Z. He, J. Wang and M. J. Pérez-Jiménez, Fault diagnosis of electric power systems based on fuzzy reasoning spiking neural P systems, *IEEE Transactions on Power Systems* **30**(3) (2015) 1182–1194.
52. A. Q. Yousif Yahya and A. Yahya, Power transformer fault diagnosis using fuzzy reasoning spiking neural P systems, *Journal of Intelligent Learning Systems and Applications* **8** (2016) 77–91.
53. M. Ionescu and D. Sburlan, Some applications of spiking neural P systems, *Computing and Informatics* **27**(3) (2008) 515–528.
54. T. Song, P. Zheng, M. D. Wong and X. Wang, Design of logic gates using spiking neural P systems with homogeneous neurons and astrocytes-like control, *Information Sciences* **372** (2016) 380–391.
55. C. Diaz, G. Sanchez, G. Duchen, M. Nakano and

H. Perez, An efficient hardware implementation of a novel unary spiking neural network multiplier with variable dendritic delays, *Neurocomputing* **189** (2016) 130–134.
56. B. Schrauwen, M. DHaene, D. Verstraeten and J. V. Campenhout, Compact hardware liquid state machines on fpga for real-time speech recognition, *Neural Networks* **21**(2) (2008) 511 – 523.
57. D. Verstraeten, B. Schrauwen and D. Stroobandt, Reservoir computing with stochastic bitstream neurons, *In Proceedings of the 16th Annual ProRISC Workshop*, 2005, pp. 454–459.
58. M. D'Haene, M. Hermans and B. Schrauwen, Toward unified hybrid simulation techniques for spiking neural networks, *Neural Computation* **26**(6) (2014) 1055–1079.
59. E. Csuhaj-Varju, *Grammar Systems: A Grammatical Approach to Distribution and Cooperation* (Gordon and Breach, London, 1994).
60. G. Păun, Grammar systems: A grammatical approach to distribution and cooperation, *Automata, Languages and Programming: 22nd International Colloquium, ICALP 1995*, eds. Z. Fülöp and F. Gécseg (Springer, Berlin, 1995), pp. 429–443.
61. L. Pan, G. Păun, G. Zhang and F. Neri, Spiking neural P systems with communication on request, **27**(08) (2017) p. 1750042.
62. Z. Wang, L. Guo and M. Adjouadi, A generalized leaky integrate-and-fire neuron model with fast implementation method, *International Journal of Neural Systems* **24**(05) (2014) p. 1440004.
63. A. Leporati, C. Zandron, C. Ferretti and G. Mauri, On the computational power of spiking neural P systems, *International Journal of Unconventional Computing* **5**(5) (2009) 459–473.
64. M. García Arnau, D. Peréz, A. Rodríguez Patón and P. Sosík, Spiking neural P systems: stronger normal forms, *International Journal of Unconventional Computing* **5** (2009) 411–425.
65. J. Wang, H. J. Hoogeboom, L. Pan, G. Păun and M. J. Pérez-Jiménez, Spiking neural P systems with weights, *Neural Computation* **22**(10) (2010) 2615–2646.
66. T. Wu, A. Păun, Z. Zhang and L. Pan, Spiking neural P systems with polarizations, *IEEE Transactions on Neural Networks and Learning Systems* (2017).
67. G. Păun, G. Rozenberg and A. Salomaa, *The Oxford handbook of Membrane Computing* (Oxford University Press, New York, 2010).
68. G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages* (Springer-Verlag, Berlin, 1997).
69. J. E. Hopcroft, R. Motwani and J. D. Ullman (eds.), *Introduction to Automata Theory, Languages, and Computation (third edition)* (Addison Wesley, Pearson Education India, New Jersey, 2001).
70. A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society* **2**(42) (1936) 230–265.
71. M. Minsky, *Computation: Finite and Infinite Machines* (Prentice–Hall, Englewood Cliffs, N.J., 1967).
72. I. Korec, Small universal register machines, *Theoretical Computer Science* **168**(2) (1996) 267–301.
73. F.-D. Bîlbîe and A. Păun, Universality of spiking neural p systems with communication on request using one type of spike, *In Proceedings of the 18th International Conference on Membrane Computing (CMC 2017)*, (Bradford, UK, July 25–28, 2017).
74. H. T. Siegelmann and E. D. Sontag, On the computational power of neural nets, *Journal of Computer and System Sciences* **50**(1) (1995) 132–150.
75. A. Gupta and L. N. Long, Character recognition using spiking neural networks, *In Proceedings of the 2007 International Joint Conference on Neural Networks*, (Orlando, Florida, USA, August 12–17, 2007), pp. 53–58.